ĐẠI HỌC QUỐC GIA TP HCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC HÀNH

LAB: 5

ĐỒNG BỘ HÓA TIẾN TRÌNH, TIỂU TRÌNH

Lớp thực hành môn: HỆ ĐIỀU HÀNH

– **IT007.M11.CLC.1**

Sinh viên thực hiện:

MSSV: 20520322

Họ và tên: Nguyễn Thị Mỹ Trân

5.4 Hướng dẫn thực hành

1. Hiện thực hóa mô hình trong ví dụ 5.3.1.2, tuy nhiên thay bằng điều kiện sau: sells <= products <= sells + [2 số cuối của MSSV + 10]

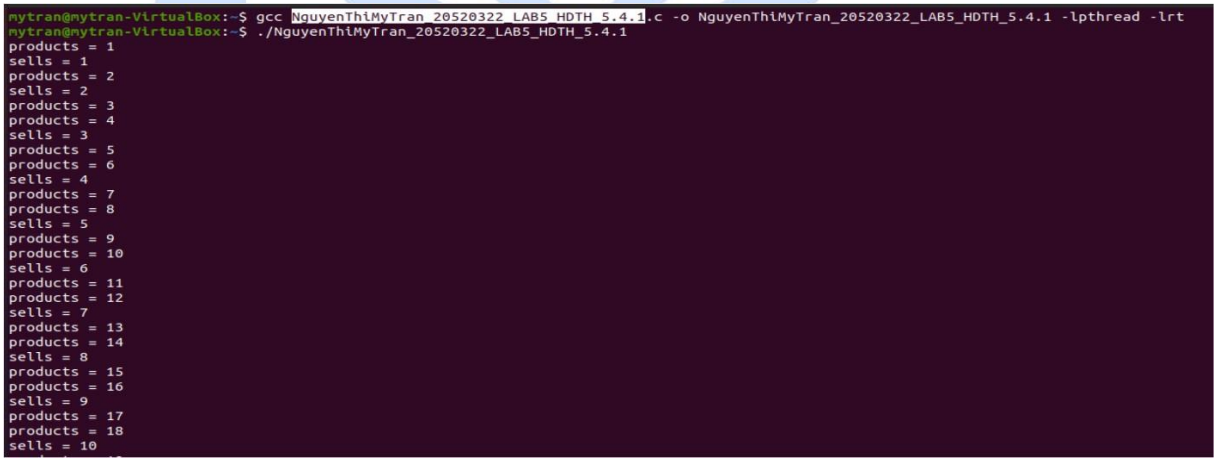
Viết chương trình lưu tại file:

NguyenThiMyTran_20520322_LAB5_HDTH_5.4.1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t sem;
int sells=0, products=0;
void* PROCESSA()
{
        while(1)
        {
                sem_wait(&sem);
                sells++;
                printf("sells = %d\n", sells);
                sleep(2);
        }
}
void* PROCESSB()
{
        while(1)
        {
```

```
                if (products <= sells + 22 + 10) {
                        products++;
                        printf("products = %d\n", products);
                        sem_post(&sem);
                        sleep(1);
                }

        }
}
void main()
{
        sem_init(&sem,0,0);
        pthread_t th1, th2;
        pthread_create(&th1, NULL, &PROCESSA, NULL);
        pthread_create(&th2, NULL, &PROCESSB, NULL);
        while(1);
}
```

Kết quả chạy chương trình:



2. Cho một mảng a được khai báo như một mảng số nguyên có thể chứa n phần tử, a được khai báo như một biến toàn cục. Viết chương trình bao gồm 2 thread chạy song song:

- Một thread làm nhiệm vụ sinh ra một số nguyên ngẫu nhiên sau đó bỏ vào a. Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi thêm vào.

- Thread còn lại lấy ra một phần tử trong a (phần tử bất kỳ, phụ thuộc vào người lập trình). Sau đó đếm và xuất ra số phần tử của a có được ngay sau khi lấy ra, nếu không có phần tử nào trong a thì xuất ra màn hình "Nothing in array a".

- Chạy thử và tìm ra lỗi khi chạy chương trình trên khi chưa được đồng bộ. Thực hiện đồng bộ hóa với semaphore.

File code khi chưa có semaphore:

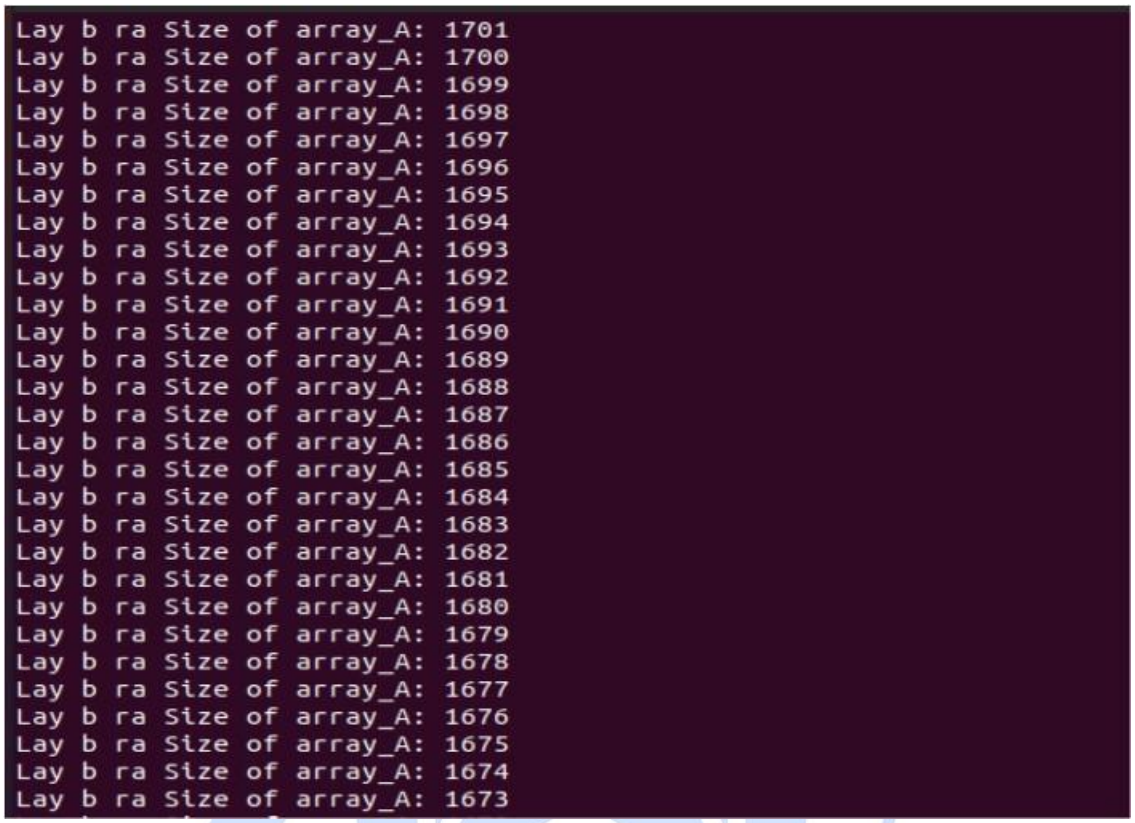
NguyenThiMyTran_20520322_LAB5_HDTH_5.4.2_NoSemaphore.c

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <time.h>
#include <stdlib.h>

pthread_t tid[2];

int a[100000];
int size_a = 0;

void* f1()
{
```

```
	while (1) {
		if (size_a > 0) {
			size_a--;
			printf("Lay b ra Size of array_A: %d\n", size_a);
		}
		else
			printf("Nothing in array_A\n");
	}
}
void* f2()
{
	while (1) {
		a[size_a] = rand() % 100000;
		size_a++;
		printf("Size of array_A: %d\n", size_a);
	}
}

int main(int argc, char* argv[])
{
	srand(time(NULL));
	int i;
	for (i = 0; i < 100000; i++) {
		a[i] = 0;
	}

	pthread_create(&tid[0], NULL, f1, NULL);
	pthread_create(&tid[1], NULL, f2, NULL);

	for (i = 0; i < 2; i++)
```

```
        pthread_join(tid[i], NULL);
}
```

Kết quả chạy chương trình khi chưa có semaphore:



Vì chưa có semaphore nên chương trình chưa được đồng bộ (Lỗi logic dùng semaphore để xử lý lỗi logic. B lấy ra kích thước của mảng A khi A chưa được vào chương trình.

File code khi có semaphore:

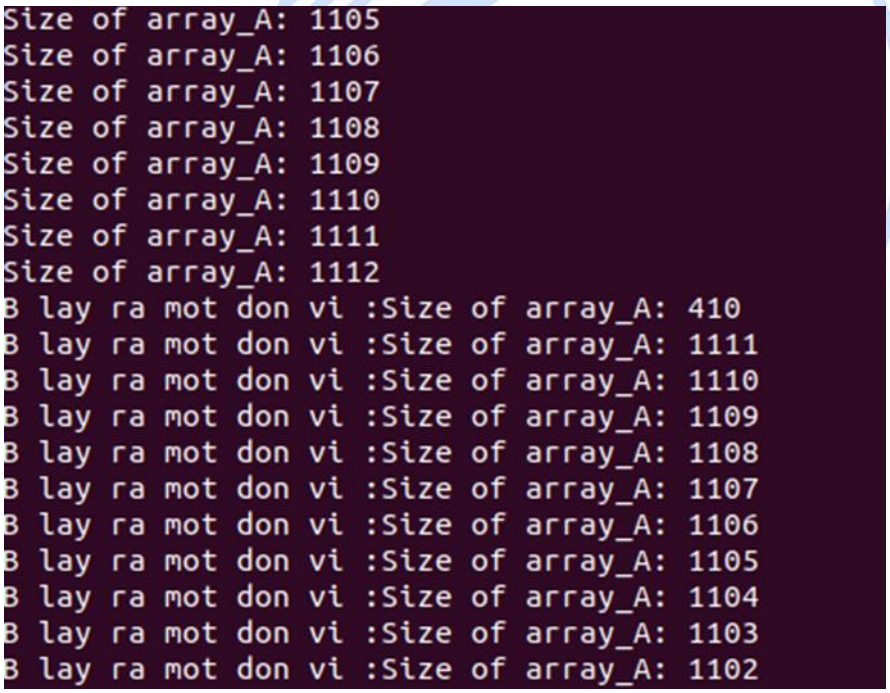
NguyenThiMyTran_20520322_LAB5_HDTD_5.4.2_Semaphore.c

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <time.h>
#include <stdlib.h>
pthread_t tid[2];
```

```
sem_t sem;

int a[100000];
int size_a = 0;

void* f1()
{
	while (1) {
		sem_wait(&sem);
		if (size_a > 0) {
			size_a--;
			printf("Size of array_A: %d\n", size_a);
		}
		else
			printf("Nothing in array_A");
	}
}
void* f2()
{
	while (1) {
		a[size_a] = rand() % 100000;
		size_a++;
		printf("Size of array_A: %d\n", size_a);
		sem_post(&sem);

	}
}

int main(int argc, char* argv[])
{
	srand (time(NULL));
```

```
    int i;
    for (i = 0; i < 100000; i++) {
        a[i] = 0;
    }

    sem_init(&sem, 0, 0);
    pthread_create(&tid[0], NULL, f1, NULL);
    pthread_create(&tid[1], NULL, f2, NULL);
    for (i = 0; i < 2; i++)
        pthread_join(tid[i], NULL);
}
```

Kết quả chương trình khi có semaphores:



3.Thực hiện process A và Process B chạy song song:

Viết chương trình lưu tại file:

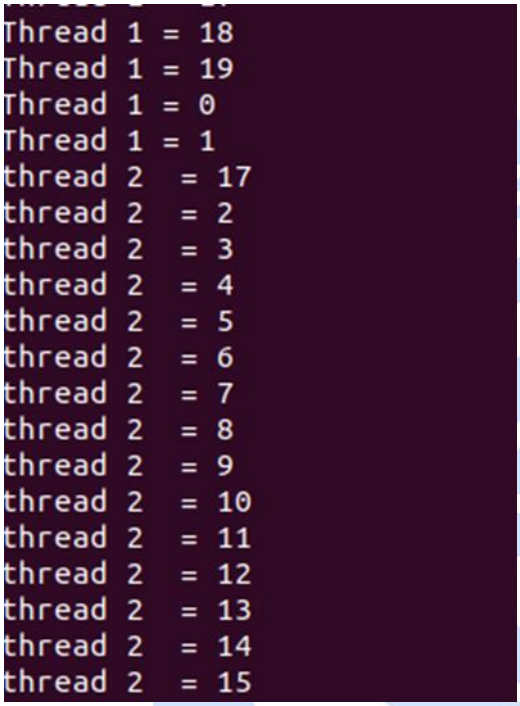
NguyenThiMyTran_20520322_LAB5_HDTH_5.4.3.c

```
#include <stdio.h>
```

```
#include <semaphore.h>
#include <pthread.h>

pthread_t tid[2];

sem_t sem;
int x = 0;

void* f1()
{
     while (1) {
          x = x + 1;
          if (x == 20)
               x = 0;
          printf("Thread 1 = %d\n", x);
     }
}
void* f2()
{
     while (1) {
          x = x + 1;
          if (x == 20)
               x = 0;
          printf("Thread 2= %d\n", x);
     }
}

int main(int argc, char* argv[])
{
     int i;
     sem_init(&sem, 0, 0);
```

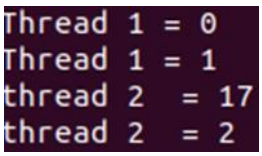
```
    pthread_create(&tid[0], NULL, f1, NULL);
    pthread_create(&tid[1], NULL, f2, NULL);
    for (i = 0; i < 2; i++)
        pthread_join(tid[i], NULL);
}
```

Kết quả chạy chương trình



Process thứ nhất chạy được từ 1 đến19 và khi đến if(x==20) thì sẽ quay lại giá trị 0. Và trong khi process 1 đang chuẩn bị đọc dữ liệu từ đĩa, thao tác đọc chưa hoàn tất thì bị process 2 ghi đè dữ liệu mới lên dữ liệu cũ.

Nguyên nhân: Khi chạy song song 2 process thứ nhất và thứ hai, biến toàn cục x đều được cho cả hai process biến x gọi ra nhưng không có semaphore để báo rằng đây là hai tiến trình dùng chung biến x dẫn đến việc xảy ra đụng độ khi truy cập và xử lí biến chung như dòng trên.



Kết luận: Kết quả sai với yêu cầu.

Nhận xét: Chương trình chạy bất hợp lý.

4. Đồng bộ với mutex để sửa lỗi bất hợp lý trong kết quả của mô hình Bài 3.

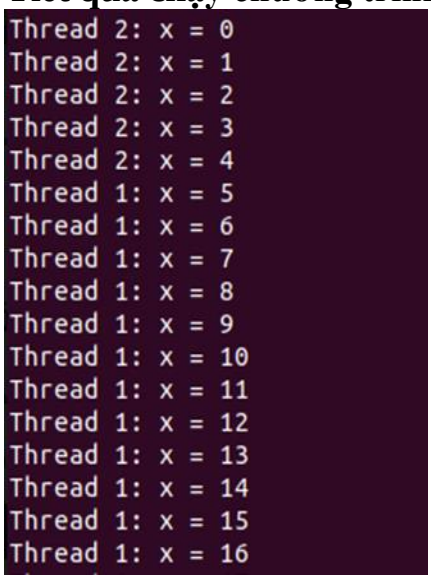
Viết chương trình lưu tại file:

NguyenThiMyTran_20520322_LAB5_HDTH_5.4.4.c

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>

pthread_t tid[2];
pthread_mutex_t mutex;
sem_t sem;
int x = 0;

void* f1()
{
     while (1) {
          pthread_mutex_lock(&mutex);
          x = x + 1;
          if (x == 20)
               x = 0;
          printf("Thread 1: x = %d\n", x);
```

```
            pthread_mutex_unlock(&mutex);
      }
}
void* f2()
{
      while (1) {
            pthread_mutex_lock(&mutex);
            x = x + 1;
            if (x == 20)
                  x = 0;
            printf("Thread 2: x = %d\n", x);
            pthread_mutex_unlock(&mutex);
      }
}

int main(int argc, char* argv[])
{
      int i;
      sem_init(&sem, 0, 0);
      pthread_mutex_init(&mutex, NULL);
      pthread_create(&tid[0], NULL, f1, NULL);
      pthread_create(&tid[1], NULL, f2, NULL);
      for (i = 0; i < 2; i++)
            pthread_join(tid[i], NULL);

      pthread_mutex_destroy(&mutex);
      return 1;
}
```

Kết quả chạy chương trình:



Kết luận: Kết quả được thực thi đúng với yêu cầu đồng bộ hóa tiến trình

5.5 Bài tập ôn tập

1.Biến ans được tính từ các biến x1, x2, x3, x4, x5, x6 như sau:

w = x1 * x2; (a)

v = x3 * x4; (b)

y = v * x5; (c)

z = v * x6; (d)

y = w * y; (e)

z = w * z; (f)

ans = y + z; (g)

Giả sử các lệnh từ (a) → (g) nằm trên các thread chạy song song với nhau. Hãy lập trình mô phỏng và đồng bộ trên C trong hệ điều hành Linux theo thứ tự sau:

(c), (d) chỉ được thực hiện sau khi v được tính

(e) chỉ được thực hiện sau khi w và y được tính

(g) chỉ được thực hiện sau khi y và z được tín

Viết chương trình lưu tại file: NguyenThiMyTran_20520322_LAB5_Bai1.c

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>
#include <unistd.h>

sem_t p1_5, p1_6, p2_3, p2_4, p3_5, p4_6, p5_7, p6_7;

int x1 = 1;

int x2 = 2;

int x3 = 3;

int x4 = 4;

int x5 = 5;

int x6 = 6;

int w, v, z, y, x;

int ans = 0;

void* PROCESS1()

{

        w = x1 * x2;

        printf("w = %d\n", w);

        sem_post(&p1_5);
```

```
        sem_post(&p1_6);
        sleep(1);
}
void* PROCESS2()
{
        v = x3 * x4;
        printf("v = %d\n", v);
        sem_post(&p2_3);
        sem_post(&p2_4);
        sleep(1);
}
void* PROCESS3()
{
        sem_wait(&p2_3);
        printf("y = %d\n", y);
        y = v * x5;
        sem_post(&p3_5);
        sleep(1);
}
void *PROCESS4()
{
        sem_wait(&p2_4);
        printf("z = %d\n", z);
        z = v * x6;
        sem_post(&p4_6);
        sleep(1);
}
void *PROCESS5()
{
        sem_wait(&p1_5);
```

```
        sem_wait(&p3_5);
        y = w * y;
        printf("y = %d\n", y);
        sem_post(&p5_7);
        sleep(1);
}
void *PROCESS6()
{
        sem_wait(&p1_6);
        sem_wait(&p4_6);
        z = w * z;
        printf("z = %d\n", z);
        sem_post(&p6_7);
        sleep(1);
}
void* PROCESS7()
{
        sem_wait(&p5_7);
        sem_wait(&p6_7);
        ans = y + z;
        printf("ans = %d\n", ans);
        sleep(1);
}
void main()
{
        sem_init(&p1_5, 0, 1);
        sem_init(&p1_6, 0, 0);
        sem_init(&p2_3, 0, 0);
        sem_init(&p2_4, 0, 0);
        sem_init(&p3_5, 0, 0);
        sem_init(&p4_6, 0, 0);
```

```
        sem_init(&p5_7, 0, 0);
        sem_init(&p6_7, 0, 0);
        pthread_t th1, th2, th3, th4, th5, th6, th7;
        pthread_create(&th1, NULL, &PROCESS1, NULL);
        pthread_create(&th2, NULL, &PROCESS2, NULL);
        pthread_create(&th3, NULL, &PROCESS3, NULL);
        pthread_create(&th4, NULL, &PROCESS4, NULL);
        pthread_create(&th5, NULL, &PROCESS5, NULL);
        pthread_create(&th6, NULL, &PROCESS6, NULL);
        pthread_create(&th7, NULL, &PROCESS7, NULL);
        while (1);
}
```

Kết quả chạy chương trình:

```
mytran@mytran-VirtualBox:~$ gcc NguyenThiMyTran_20520322_LAB5_Bai1.c -o NguyenThiMyTran_20520322_LAB5_Bai1 -lpthread -lrt
mytran@mytran-VirtualBox:~$ ./NguyenThiMyTran_20520322_LAB5_Bai1
w = 2
v = 12
y = 0
z = 0
y = 120
z = 144
ans = 264
```

HẾT